

AD-A273 701



2

WL-TR-93-3098

EXPERT SYSTEM RULE-BASE EVALUATION
USING REAL-TIME PARALLEL PROCESSING



JAMES L. NOYES

SEPTEMBER 1993

FINAL REPORT FOR 06/01/93-08/01/93



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7562

SPF 93-30217

93 12 13 067

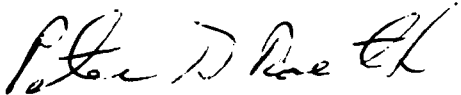
**Best
Available
Copy**


NOTICE


When government drawings, specifications, or other data are used for any purpose other than in connection with a definitely government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the government may have formulated, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise in any manner construed, as licensing the holder or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related there to.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.


PETER G. RAETH, MAJOR, USAF
CHIEF, PILOT/VEHICLE INTERFACE
TECHNOLOGY BRANCH
WRIGHT LABORATORY


TIMOTHY G. KINNEY, LT COL, USAF
DEPUTY, COCKPIT INTEGRATION DIVISION
WRIGHT LABORATORY


PAUL E. BLATT
CHIEF, COCKPIT INTEGRATION DIVISION
WRIGHT LABORATORY

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify Wright Laboratory; Flight Dynamics Directorate; WL/FIPA Bldg 146; 2210 Eighth Street Ste 1; Wright-Patterson Air Force Base, OH 45433-7511 USA

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE SEP 1993	3. REPORT TYPE AND DATES COVERED FINAL 06/01/93--08/01/93		
4. TITLE AND SUBTITLE EXPERT SYSTEM RULE-BASE EVALUATION USING REAL-TIME PARALLEL PROCESSING		5. FUNDING NUMBERS PE 62201 PR 2402 TA 04 WU 86		
6. AUTHOR(S) JAMES L. NOYES PROFESSOR OF COMPUTER SCIENCE DEPARTMENT OF COMPUTER SCIENCE WITTENBERG UNIVERSITY				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) FLIGHT DYNAMICS DIRECTORATE WRIGHT LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT PATTERSON AFB OH 45433-7562		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) FLIGHT DYNAMICS DIRECTORATE WRIGHT LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT PATTERSON AFB OH 45433-7562		10. SPONSORING / MONITORING AGENCY REPORT NUMBER WL-TR-93-3098		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) A large rule-based expert system with each rule involving perhaps 10 out of 100,000 possible Boolean criteria, can require a significant amount of processing time to evaluate. This time can be reduced if all rules have a single consequent and have antecedents that contain only conjunctions of the Boolean criteria or their complements. If the consequents do not insert new facts into the rule-base, then parallel processing can be used with great efficiency. The value of a rule-based expert system to help solve a variety of diagnostic and advisory needs has been well-demonstrated over the last 2 decades. Parallel processing has become increasingly important for embedded systems in order to accelerate a variety of computations. This report discusses research connected to the development of a data structure and algorithm to perform parallel inferencing in rule-based systems. It also discusses a simulation technique for estimating the number of processors needed to evaluate a given number of rules and criteria within the required time.				
14. SUBJECT TERMS EXPERT SYSTEMS, KNOWLEDGE-BASED SYSTEMS, RULE-BASED SYSTEMS, ARTIFICIAL INTELLIGENCE, REAL-TIME PROCESSING, PARALLEL PROCESSING, DISTRIBUTED COMPUTING, DECISION SUPPORT, COCKPIT AUTOMATION, FUNCTION ALLOCATION, SIMULATION			15. NUMBER OF PAGES 33	
16. PRICE CODE				
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

1. INTRODUCTION	1
2. EXPERT SYSTEM FORMULATION	3
3. DATA STRUCTURE AND ALGORITHM DEVELOPMENT	6
4. METHOD-1	8
5. METHOD-2	13
6. DEALING WITH UNCERTAINTY	16
7. SIMULATION GUIDELINES AND RESULTS	19
8. TEST RESULTS AND CONCLUSIONS	28
9. REFERENCES	30

List of Figures

1. Basic Systems Diagram of Parallel Real-Time Rule-Based Expert System	15
2. Rule Simulation Program Sample Screen Output ...	23

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input checked="checked" type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

ACKNOWLEDGEMENTS

First, I would like to thank Major Peter G. Raeth, Chief of the Pilot/Vehicle Interface Technology Branch (WL/FIPA) and my research advisor during this 10-week period. His guidance was very clear and he went out of his way to see that the problem was well defined. He helped me obtain the related literature and computing resources. Next I would like to thank Lt Col. Tim Kinney, Deputy of the WL/FIP Division, and Dr. Jim Olsen, Chief Scientist of WL/FI for their invitation to do this type research at Wright Laboratory. Finally, I would like to thank Andy Probert, WL/FIPC, for his insights into what pilots really expect from an on-board expert system.

Editor's Note: Dr Noyes performed this research as part of the Summer Faculty Research Program sponsored by the Air Force Office of Scientific Research, Bolling Air Force Base, Washington, DC.

EXPERT SYSTEM RULE-BASE EVALUATION USING REAL-TIME PARALLEL PROCESSING

James L. Noyes

1. INTRODUCTION

The value of a rule-based expert system (ES) to help solve a variety of diagnostic and advisory needs has been well-demonstrated over the last 2 decades, as discussed by Noyes in [1]. Sometimes, a large number, say $O(10^3)$, of the ES rules must be continuously checked in real-time (e.g., every $O(10^{-1})$ seconds) due to stringent requirements imposed by the problem. In addition, while each rule may use only $O(10^1)$ criteria, there may be a very large number of possible criteria, say $O(10^5)$, for the entire rule-base that must be checked during each time-step. Because of these timing demands, parallel processing may be deemed necessary. Parallel processing has become increasingly important in order to accelerate a variety of computations, as discussed by Noyes in [1] and Trippi and Turban in [2]. This report discusses research connected to the development of a data structure and an algorithm to evaluate this type of rule-base and the estimation of the processor speeds necessary to evaluate these rules within the required time. The particular application for this real-time ES is a rule-base to aid the pilot of modern fighter or transport aircraft and the remainder of this report will address this application. However, the results of the research presented here could be used in other applications.

All assumptions in this report on sensor update rates, number of sensors, and other matters that determine real-time response are based on conversations with F-4 pilots and program managers working aircraft systems applications. The numbers are application-specific. The actual numbers for the application at hand and the configuration of the available processing hardware determine system ability to respond in real-time. The simulation and inferencing methods developed in this report are designed to enable system expandability to ensure real-time performance.

Throughout this report the notation " $O(x)$ " is found. This means "on the order of x ." The notation " $O(10^1)$ " means "on the order of 10" or "somewhere in the neighborhood of 10."

2. EXPERT SYSTEM FORMULATION

This ES rule-base formulation depends upon a state vector, a criteria vector, a response (action) vector, and a set of rules.

The aircraft state vector \mathbf{s} consists of z continuous and discrete components (state variables) completely describing the state of the aircraft at a given time-step t_k , of magnitude $O(10^{-1})$ seconds. These values are determined by a collection of on-board sensors and there may be $O(10^2)$ of these. For example, state variable s_{12} might represent the number of pounds of fuel currently in the fuel cells. The aircraft criteria vector \mathbf{c} is a vector of m Boolean (True or False) variables. Each of these variables is based upon a value of one or more of the variables in the state vector. For example, criterion c_{33} might represent the relationship between the current amount of fuel and a minimum fuel reserve (e.g., $c_{33} = [s_{12} \leq f_R]$) and c_{33} is True when there is insufficient fuel reserve.

A set of n rules, of order $O(10^3)$, defines the on-board expert system that will advise the pilot and, with the pilot's consent, act on his or her behalf. Each ES rule can be formulated in terms of a conjunction of simple Boolean criteria that lead to a single action. If all of a given rule's criteria are true (based upon the elements of the corresponding criteria vector), an action will result. This action could either be an activity that is

automatically performed for the pilot or it could be a recommendation to the pilot. All of these actions define an action vector \mathbf{a} of size p . Each rule is expected to involve only a relatively small number of m possible criteria. For this report, m is of order $O(10^5)$. For example, each rule may have up to 10 criteria. The rule-base is built off-line, and not modified during the search process. For example, a typical rule might look like this ("~" means "NOT"):

Rule R_{123} : $\text{action}_{12} \leq c_1 \ \& \ \sim c_5 \ \& \ c_6 \ \& \ c_{18} \ \& \ \sim c_{47} \ \& \ \sim c_{99}$

This rule is interpreted as stating that action_{12} will be taken if c_1 , c_6 , and c_{18} are all true while c_5 , c_{47} , and c_{99} are all false.

In a typical ES, the inference engine performs three standard operations: the match operation matches the criteria against the rules to see which could fire, the resolve operation chooses which of these rules will fire, and the execute operation actually fires these rules and updates working memory. For the given problem, these operations can be simplified into a simple match-fire operation with no resolution operation nor updates.

While it is assumed that no action alters the criteria vector \mathbf{c} in any way at any time-step t_k , it is possible that different rules can have the same action. Hence, by expressing each rule only in terms of simple AND and NOT logic, its evaluation can be done very efficiently and independently. (Note that OR constructs

are equivalent to multiple rules that specify the same action.)

Duplicate actions are prevented by the action triggering mechanism that is external to the inference engine described in this report. This mechanism sets a "triggered" flag when the action is started. In a given update cycle, this flag can only be set once. All other attempts to set this flag are ignored. When the action is completed, the flag is reset.

Conflicting actions can be resolved by expanded criteria such as " $\sim a_{51}$." This means that the rule's consequent action would not take place if action #51 is underway. This technique would mean that the blackboard would have to be updated for each action start and completion. Thus, the action vector would have an associated "action triggered" vector. This could be accomplished by simply making the element in the action vector negative for "action underway" or positive for "action not underway."

3. DATA STRUCTURE AND ALGORITHM DEVELOPMENT

The data structure and algorithms developed to evaluate this ES, are designed to be used by a single fast processor or by parallel processors that can have a correspondingly slower clock-speed. This data structure utilizes the notion of a blackboard that contains the state and criteria vectors described above. In addition, three other vectors, the action, query, and index vectors completely define the rule-base. Unlike the s and c vectors, these three vectors are not updated, and can reside either in the blackboard or some other data storage area. The query vector contains a list of the criteria for each rule. The index vector elements point to the criteria that apply to each rule.

A blackboard is a global and dynamic data base for the communication of independent asynchronous knowledge sources for related aspects of a given problem. The aircraft system blackboard will contain the state vector s and criteria vector c . These vectors will be updated by an independent on-board computer (not involved in the rule search) at each time-step. Each update of the vector c will immediately initiate a new evaluation of the rules' criteria, so the rule search must be complete for the criteria vector at time-step t_k before the criteria vector is updated at time-step t_{k+1} . Hence, this blackboard must also be accessible by the computer that executes the rule processing algorithm. Criteria vector updates are discussed by Raeth in [3, 4].

While it is possible that a fast single processor computer could be used on the aircraft, the most likely hardware configuration for the rule processing algorithm will involve eight (8) parallel processors. This number is convenient since systems developers typically place eight processors on one board for embedded applications. (Transputers are a likely candidate since they are currently available to the sponsor.)

One of the eight processors will serve as the combined master and I/O processor and will have one of its four serial I/O ports connected to the common data bus on the aircraft. This processor will accept the criteria vector and possible pilot input and provide the ultimate rule search output. The remaining 7 processors may use each of their four I/O ports to connect to any other processor. A preset architecture will be employed. This architecture can be as simple as a ring or as complex as a mesh.

Both algorithms presented in this report should be considered as prototypes and have been implemented in Pascal. If a particular algorithm is sufficiently successful, it will eventually be implemented in Ada or Ada-9X. This will permit a ready transition to operational aircraft since the Ada standard has been mandated for DoD embedded applications. Mil-Std versions of transputers exist. Together, transputers programmed in Ada represent a mature and installable parallel processing capability that takes advantage of modern processor architectures.

4. METHOD-1

The simplest method for this ES evaluation assumes that the if-then-action rules and their criteria are listed in priority order. This is equivalent to a priority-oriented backward chaining method. This is the obvious choice when $n \ll m$ and no other assumptions are made about available data. (Note that if these rules were not prioritized, then this first algorithm could be viewed as a forward chaining algorithm.) Because no OR-logic is present in a given rule, the current rule-processor should stop with the first $c_i = \text{False}$ (or first $c_i = \text{True}$ in the case of $\sim c_i$). If these rules were ranked and evaluated from highest to lowest priority, then the first action produced (if any) would be the most important from the pilot's point of view. If required, different levels of parallelism could be employed during this evaluation process. If the processing time is not fast enough, then rules having the same priority could be grouped according to their number of criteria in order to equalize the work among the parallel processors, as discussed by Tout and Evans in [5]. A simple example of a rule-base with four rules is:

Rule R_1 : $\text{action}_1 \Leftarrow c_1 \ \& \ c_3 \ \& \ \sim c_4 \ \& \ c_{40} \ \& \ \sim c_{98} \ \& \ c_{99}$
Rule R_2 : $\text{action}_1 \Leftarrow c_2 \ \& \ c_4 \ \& \ c_{22} \ \& \ \sim c_{85}$
Rule R_3 : $\text{action}_2 \Leftarrow c_5 \ \& \ c_{99}$
Rule R_4 : $\text{action}_3 \Leftarrow c_1 \ \& \ c_{50}$

Note that R_1 is the highest priority rule and R_4 is the lowest priority rule. The criteria are evaluated left to right. Evaluation stops as soon as a False is detected. The left-to-right evaluation can be thought of as assuming that the left-most

criteria are expected to occur most often and are thus evaluated first.

These rules could be represented efficiently by using three vectors: the previously discussed action vector **a** whose elements each point to a specific task to be completed, a query vector q, identifying which criteria have to be checked, and an index vector End, that delimits the criteria that appear in each of the rules. For the above rule-base, consider:

RULE 1: ACTION₁ = A₁; Q₁ = 1, Q₂ = 3, Q₃ = -4, Q₄ = 40, Q₅ = -98, Q₆ = 99, so
 START₁ = 1; END₁ = 6
 RULE 2: ACTION₂ = A₁; Q₇ = 2, Q₈ = 4, Q₉ = 22, Q₁₀ = -85, so
 START₂ = 7; END₂ = 10
 RULE 3: ACTION₃ = A₂, Q₁₁ = 5, Q₁₂ = 99, so
 START₃ = 11; END₃ = 12
 RULE 4: ACTION₄ = A₃; Q₁₃ = 1, Q₁₄ = 50, so
 START₄ = 13; END₄ = 14

Here **q** employs positive integers to indicate the indices of the criteria used in the rules and negative integers for the indices of the criteria complements (NOT-criteria). Note also that Rules 1 and 2 have the same consequent. From the previous example, one has the 14-element query vector:

q: | 1 | 3 | -4 | 40 | -98 | 99 | 2 | 4 | 22 | -85 | 5 | 99 | 1 | 50 |

This allows for direct and very fast access to the **c** vector stored on the blackboard (only one internal integer multiplication

and addition are needed to compute any cell address). If parallel processors are used, this Boolean criteria vector **c** can be accessed from the blackboard by all processors. If multicomputers are used, **c** would be communicated to the local memory of each processor and this communication time will need to be considered, according to Lester in [6]. Each processor also must use components from the query vector **q**. Note the relationship $Start_{j+1} = End_j + 1$ with $Start_1 = 1$, so only the **End** unsigned integer index vector is actually needed by the algorithm. In this example, one has:

End: | 6 | 10 | 12 | 14 | which implies **Start:** | 1 | 7 | 11 | 13 |

Note that vector **q** has a number of elements equal to the sum of the number of criteria queried by each rule. Vector **End** has *n* elements, the total number of rules.

This method yields Algorithm-1, presented below, which is a relatively simple and straightforward algorithm that can utilize these data structures.

```

Forall i := 1 to n do in parallel
  begin
    if i = 1
      then j := 1
      else j := Endi-1 + 1;
    Fired := TRUE;
    while j ≤ Endi and Fired do
      begin
        k := qj;
        if k ≥ 0 and not ck then
          Fired := FALSE
        else if k < 0 and c-k then
          Fired := FALSE;
        j := j + 1
      end;
    if Fired then perform action aj
  end

```

In Algorithm-1, the Forall statement creates up to n parallel processes. If p is the number of parallel processors and $p \geq n$, then this loop completes as soon as the slowest of these processes has finished execution. Here the total parallel processing time at a given time-step is the maximum of these times. If $p < n$, then the next available processor would evaluate the next unprocessed rule, hence the total parallel processing time at a given time-step is then the maximum of all sums of the individual processor times. Notice that this reduces to a normal sequential processing algorithm when $p = 1$.

For example, if $p \geq 4$ and it takes an estimated average of 50 microseconds to check each criterion in the previous 4-rule situation, then 4 copies of the loop body will be created on 4 different processors, each with its own value of the loop control i -variable. These will execute in parallel with respective times of 300, 200, 100, 100 microseconds, at most (as soon as a FALSE is determined, the process stops for the current rule). This would then take at most 300 microseconds in parallel versus at most 700 microseconds if done sequentially, giving a speedup of $7/3$ or approximately 2.3. Here the action performance time (e.g., displaying an information screen) was not considered, nor was processor-assignment overhead or communication time. Of course, any of these three times can have a significant effect on this ES evaluation process.

5. METHOD-2

The previous method does not take advantage of searching in any informed way whenever a state variable (and hence a criterion) changes, because the indexing is in the opposite direction from rule to criterion. A second, combined forward-backward chaining method, could be used to check only the rules whose criteria values have changed since the last evaluation of the rule-base. To do this, one could also index in the opposite direction, checking only the rules having newly changed (currently "active") criteria. The forward phase identifies the changed criteria and rules that use these criteria. The backward phase is the same as before with presumably fewer rules to process. For example, using the same four rules as before, one could have something like:

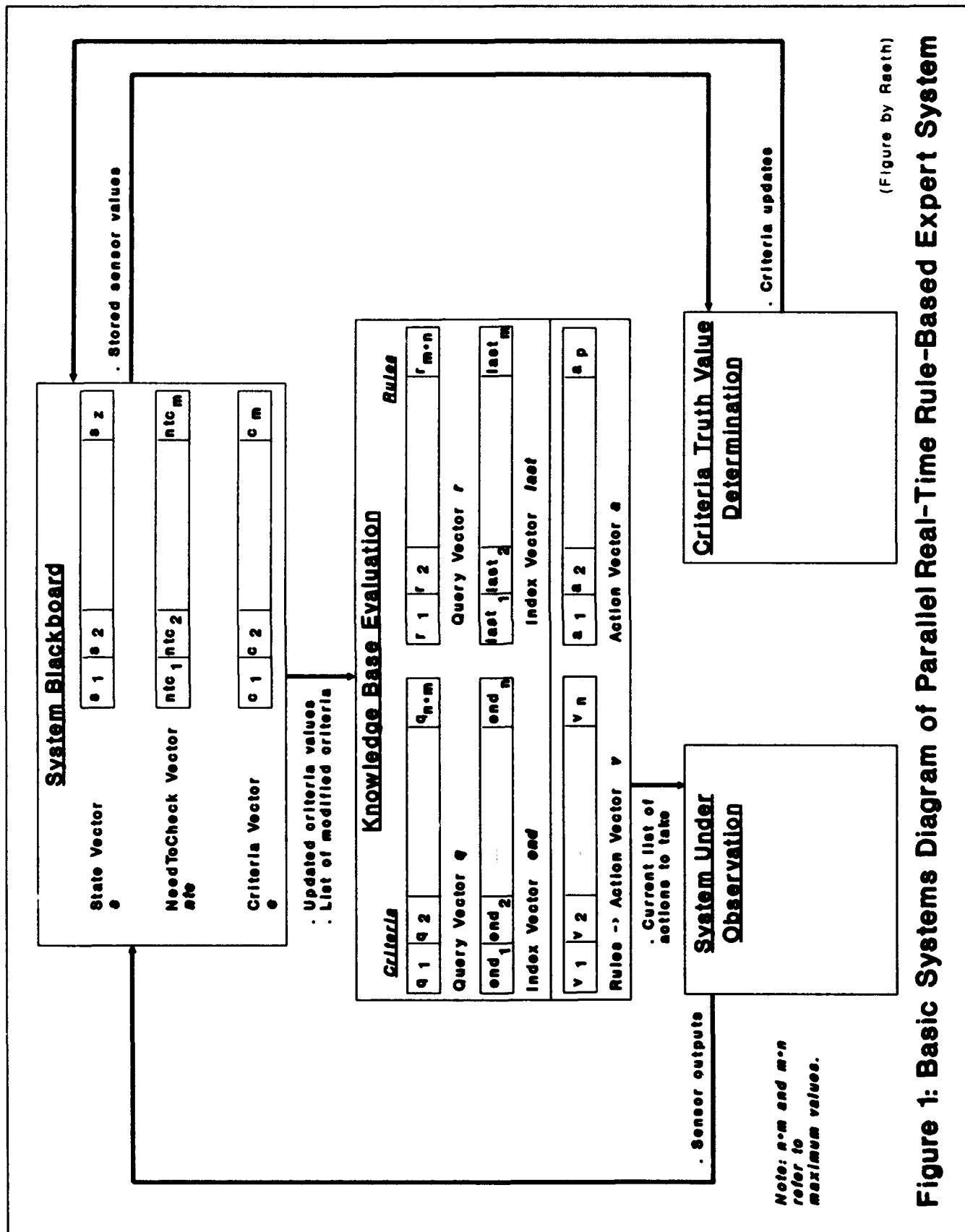
```
CRITERION C1: NEEDTOCHECK1 = FALSE; FIRST1 = 1; LAST1 = 2; R1 = 1, R2 = 4
CRITERION C2: NEEDTOCHECK2 = TRUE; FIRST2 = 3; LAST2 = 3; R3 = 2
CRITERION C3: NEEDTOCHECK3 = FALSE; FIRST3 = 4; LAST3 = 4; R4 = 1
CRITERION C4: NEEDTOCHECK4 = TRUE; FIRST4 = 5; LAST4 = 6; R5 = 1, R6 = 2
CRITERION C5: NEEDTOCHECK5 = FALSE; FIRST5 = 7; LAST5 = 7; R7 = 4
CRITERION C6: NEEDTOCHECK6 = TRUE; FIRST6 = 0; LAST6 = 0;
      NOT IN ANY RULE
.....
CRITERION C99: NEEDTOCHECK99 = FALSE; FIRST99 = 13; LAST99 = 14; R13 = 1, R14 = 3
```

Assuming criteria c_2 , c_4 , and c_6 were the only ones that changed (their **NeedToCheck** components would be set to True in the blackboard), the above would cause Rule₂, Rule₁, and Rule₂ to be consolidated into the set { Rule₁, Rule₂ } with the components NeedToCheck₂, NeedToCheck₄, and NeedToCheck₆ being reset back to

False. The efficiency of this method is related to the number of active (recently changed) criteria at any time-step. The number of criteria that change at any time-step is highly dependent upon the application. The fewer the criteria that change, the faster this method will be, but this method is more complex and requires both more data and storage than the previous method.

Each change in the state vector \mathbf{s} at time-step t_k can cause the status of the Boolean criteria vector \mathbf{c} (and its corresponding **NeedToCheck** vector) to change. Each criteria vector change, in turn, causes a set (or prioritized list) of rule numbers to be defined. Each rule in the set would contain at least one of the changed criteria and only the rules in this set need to be checked to see if all criteria hold. Once these rules have been identified, the actual criteria checking itself is done in the same manner as in Algorithm-1. It is possible to go further and only check the previously unsatisfied criteria in those rules. However, the additional software complexity, memory utilization, and execution time would likely exceed any savings compared to simply using Algorithm-1.

Figure 1 summarizes the previous discussion of Method-1 and Method-2. In order to map the rules to actions, each element in the Rules -> Action vector contains a pointer to an element in the Action vector. A record-oriented data structure can also be used to implement this system.



(Figure by Raeth)

Figure 1: Basic Systems Diagram of Parallel Real-Time Rule-Based Expert System

6. DEALING WITH UNCERTAINTY

In practice, one or more sensor failures may lead to undetermined (uncertain) components of the state vector s , which may lead to one or more unknown components of the criteria vector c . For any given rule, one of three situations must hold at time-step t_k : (1) all its criteria are known, (2) there are unknown criteria, but at least one of the known criteria fails to be satisfied, (3) all of the known criteria are satisfied, but there are still unknown criteria. The first two situations are easily addressed, since it can be exactly determined if the rule will fire or not (in the second case it will not fire). In the third situation, the values of the unknown criteria determine whether the rule will fire or not. Because of the possible interdependence of criteria, it is very difficult to determine any type of formal probability or level of certainty measure associated with the firing of this rule since multivariable conditional probabilities are involved. However, it is possible to report a possible action by simply keeping count of the number of criteria that are unknown for the given rule. This requires that each component of the criteria vector c have one of three values (True, False, Unknown), instead of just True or False as used in Algorithm-1. A possible action occurs if a rule's criteria are either True or Unknown.

The algorithm to do this is a variation of Algorithm-1, but is slightly more complex and takes more processing time. This is

because an additional IF-test is needed, and two additional counting operations are necessary for the reporting when one or more of the necessary c values are unknown. The reporting of the Ucount/Ncrit ratio is intended to give the pilot some measure of exactly how many unknown criteria (Ucount) exist relative to the total number of criteria (Ncrit) that are used in the given rule. For example, if there are 10 criteria in the rule and a possible action is reported with a ratio of 1/10, then the pilot might place more confidence in it than if a ratio of 7/10 was presented.

The algorithm designed to deal with this uncertainty is presented below as Algorithm-1u:

```

Forall i := 1 to n do in parallel
  begin
    if i = 1
      then j := 1
      else j := Endi-1 + 1;
    Fired := TRUE;
    Ncrit := 0;
    Ucount := 0;
    while j ≤ Endi and Fired do
      begin
        k := qj;
        if c|k| is Unknown then
          Ucount := Ucount + 1
        else if k ≥ 0 and ck is False then
          Fired := FALSE
        else if k < 0 and c-k is True then
          Fired := FALSE;
        j := j + 1;
        Ncrit := Ncrit + 1
      end;
    if Fired then
      if Ucount = 0
        then perform action aj
        else report possible aj with Ucount/Ncrit ratio
      end
  end

```

This algorithm could also be modified to report exactly which unknown criteria caused the problem. When considered in the total application context, it may also be useful to report the failed sensors that caused the unknown criteria.

7. SIMULATION GUIDELINES AND RESULTS

The software and hardware realization associated with the rule processing algorithm will depend upon the amount and frequency of the available data and the real-time constraints for the solution. To see if an algorithm is acceptable, it could be implemented within a specially written Turbo Pascal simulation program such as PASIM (Pilot's Associate Simulator). This simulator can be used to test Algorithm-1 and estimate both the sequential and parallel processing speeds. Because of the interest in handling uncertainty, the Pilots Associate Reliability Simulator (PARSIM) program was developed to test Algorithm-1u. PARSIM can be thought of as an extension of PASIM that also allows the user to incorporate an uncertainty percentage that will also simulate sensor failures throughout the flight. In this section, sample simulation results are given for both of these algorithms. However, most of the emphasis is placed upon Algorithm-1u as implemented by the PARSIM program. Both PASIM and PARSIM are designed to use a single processor computer to simulate a parallel processing system.

The current PARSIM program parameters include (1) a maximum of 10,000 rules that are assumed to be in priority order, (2) a maximum of 10,000 different actions (during a given time-step, actions can be listed for all the rules that are fired), (3) a maximum of 32,760 different criteria can be used altogether (this

is the largest single block of data allowed in Turbo Pascal and 32,767 is the largest positive integer), (4) a maximum of 8 transputer processors can be used (one of these used strictly for I/O). The use of dynamic (array) variables in the program was ultimately necessary to allow the sizes achieved above. Note that a given criteria can appear in more than one rule and that a consequent action can be triggered by more than one rule.

In order to perform an effective simulation, one needs to know the processor speed in (1) evaluating a single Boolean criterion c_i , and (2) performing any recommended action produced by a rule. The Inmos transputers to be simulated are T800-20 32-bit models with math coprocessors. These transputers have a clock-speed of 20 MHz and up to 4 megabytes of memory each. A transputer was not available in this study. However, the more critical criterion evaluation speed can be estimated or bounded by empirically timing this evaluation on the processors below (all including math coprocessors). Here are the approximate averages of the measured processing speeds required to evaluate a single criterion based upon Algorithm-1:

Intel 80386/16MHz:	81 microseconds	=	8.1×10^{-5} seconds
Intel 80486/33MHz:	16 microseconds	=	1.6×10^{-5} seconds
Intel 80486/50MHz:	12 microseconds	=	1.2×10^{-5} seconds

The 386/16 processor is the slowest of these and presumably the closest in processing speed to the T800-20 transputer. For Algorithm-1u with the 386/16 processor, the approximate average

criterion evaluation speed was found to be 115 microseconds (i.e., it takes almost 42% longer to evaluate a single criterion).

PARSIM randomly generates up to a given number of criteria for each of n rules. It also generates random Boolean values for m criteria, and updates them at random for each time-step. Many random numbers are required during a typical simulation. The built-in Turbo Pascal pseudo-random number (PRN) generator called RANDOM, did not produce a sufficient amount of PRNs, so the uniform (0,1) real full-period PRN generator (implemented with 32-bit integers) is employed. This technique is discussed by Park and Miller in [7] and by Press, et al. in [8]. This is done with the seed update $seed_{t+1} := 16807seed_t \bmod 2147483647$ and producing the uniform PRN by using $u := seed_t/2147483647$. As usual, at the start of a typical simulation, the "randomized" initial $seed_{t+1}$ is obtained from the system clock.

A nominal mission length for a fighter aircraft (such as an F-16) might range from 1-2 hours up to as many as 5 hours of flying time. If one assumes that sensor updates all occur at 0.1-second intervals, this dictates the simulation time-step. For example, a 90 minute flight would take 54,000 time-steps ($90 \times 60 \times 10$), and if there were 10,000 rules with up to 10 criteria per rule (an average of 5 criteria per rule), it would take about 18 hours to run the PARSIM code on a 486/33 machine. On that same computer, the PARSIM code takes longer, for the reasons already indicated, hence a much

shorter number of time-steps was used.

In the simulation, all rules in the rule-base are evaluated for each time-step in the flight; this is to produce the "worst-case" situation so that any necessary processor speed-up can be identified. Actions are triggered as soon as a rule fires. It is assumed that another set of processors performs the actions. Obviously the search could be significantly faster if the algorithm could terminate after the first action is triggered.

Figure 2 indicates essentially what is shown on the screen when PARSIM executes (the screen size has more columns than this page of text so the wording has been slightly modified). The underlined quantities represent the simulation input and output values. As with any simulation, these values contain a measure of uncertainty.

Specifically, Figure 2 shows the input and output of a short PARSIM simulation with 4,000 rules with up to 10 criteria per rule generated. The number of unique actions chosen does not affect the simulation and was arbitrarily chosen to be 4,000 also. The input of $1.15e-4$ indicates the estimated average time, in seconds, that it will take the on-board rule-processor to process a single criterion (typical of a 386/16). The action time is input as zero, since it is assumed that the ES triggers another computer to perform this action. No intermediate output is requested (it is only feasible to do this when the number of rules is very small).

Here 12,000 is the number of time-steps. Depending upon the sensor update time, this could correspond to different flight times. For example, if 0.1 second is the update time, then 0.1×12000 seconds corresponds to 20 minutes of flight time, but if 0.5 second is the sensor update time, the flight is 1 hour and 40 minutes. The number 16027 is the total number of unique criteria generated. When the user enters 10, it indicates that 10% of these are expected to fail before the end of the flight.

RULE SIMULATION PROGRAM

THIS SIMULATES THE REAL-TIME PROCESSING OF A SET OF EXPERT SYSTEM RULES. LOGICAL CONTRADICTIONS WITHIN THE GENERATED RULES ARE NOT GUARANTEED, NOR ARE THE ABSENCE OF DUPLICATE RULES. NEITHER SHOULD SIGNIFICANTLY EFFECT THE SIMULATION. IT IS GUARANTEED THAT THERE WILL BE NO DUPLICATE CRITERIA IN A RULE.

INPUT:

ENTER THE NUMBER OF RULES TO GENERATE (1,...,10000): 4000
 ENTER THE NUMBER OF DIFFERENT ACTIONS (1,...,10000): 4000
 ENTER THE CRITERIA LIMIT FOR EACH RULE OUT OF 32760 POSSIBLE (1,...,28761): 10
 ENTER THE SIMULATED TIME NEEDED TO EVALUATE A SINGLE CRITERION: 1.15E-4
 ENTER THE SIMULATED TIME NEEDED TO PERFORM A SINGLE ACTION: 0
 ENTER THE TOTAL NUMBER OF PARALLEL PROCESSORS (2,...,8): 8
 ENTER THE AMOUNT OF INTERMEDIATE OUTPUT DESIRED (0 IS NOMINAL)
 - NONE(0), FIRST ACTION(1), RULES & ACTIONS(2), RULES, ACTIONS & CRITERIA(3): 0
 ENTER THE (NON-NEGATIVE) NUMBER OF SIMULATION TIME-STEPS: 12000
 ENTER THE UNCERTAINTY PERCENT FOR THE 16027 CRITERIA GENERATED [0.0,100.0]: 10
 OUTPUT:

THE ACTUAL LAPSED SYSTEM CLOCK A-TIME WAS 1.499850E+0003 SECONDS, WITH 21966 CRITERIA PROCESSED AND 1723 UNIQUE RULE(S) OUT OF 4725280 FIRED.
 ON AVERAGE THERE WERE 5 CRITERIA PER RULE WITH 3.125E-0005 SECONDS NEEDED TO PROCESS EACH RULE AND 1.250E-0001 SECONDS FOR THE ENTIRE RULE-BASE.
 THE SIMULATED SEQUENTIAL S-TIME (ONE CPU) WAS 1.034389797E+0004 TIME UNITS.
 THE SIMULATED PARALLEL P-TIME (8 PROCESSORS) WAS 1.529650535E+0003 TIME UNITS.
 MAX{A-TIME}= 1.7000E-0001, MAX{S-TIME}= 9.0988E-0001, MAX{P-TIME}= 1.3720E-0001

Figure 2: Rule Simulation Program Sample Screen Output

In the output, the a-time is the total for the computer executing PARSIM to process all the criteria in all the rules for

the entire flight. (The example in Figure 2 was run on a 486/33 and took 1,499.85 seconds to do this.) Its main purpose is to help determine the overall average single criterion evaluation time for the particular computer being used. If the user enters a 1 for the single criterion evaluation time, then the s-time (simulated sequential time) is simply a count of the number of criteria processed. By dividing this count into the a-time, one obtains the average time to process a single criterion for the processor on the computer currently being used. By making several runs of this type (e.g., 15 runs), one can obtain a reasonable estimate of this overall average.

The key output values are the last two given in Figure 2. They represent the maximum simulated sequential and parallel times over the entire flight that it will take to evaluate the rule-base. From these two values, it can be determined if the entire rule-base can be processed in less time than the sensor update time. For example, if the sensor updates are done every 0.1 second, then the rule processing time at any time-step must not be slower than this. Here a single processor takes slightly over 0.9 second to process the entire rule-base with the sequential form of Algorithm-1 while the 8-processor parallel version of the same algorithm takes slightly over 0.1 second. For this single simulation, neither the sequential nor the 8 parallel processors (only 7 actually processing the rules) will process the rule-base in an acceptable amount of time. However, if the sensor update time is 0.5 second,

then the parallel processing is fast enough since $0.1372 < 0.5$, but the sequential processing is still not fast enough. If this algorithm is implemented on multicomputers that have no common memory, then updated data, such as the **NeedToCheck** vector, will have to be communicated to each local rule-processor for each sensor update cycle. This takes an additional amount of time. For example, if this time were 0.2 second, the parallel processing is still fast enough since 0.3372 is still less than 0.5.

Of course, conclusions such as the above should not be based upon just one simulation run. One should run several simulations, at least 10, with the same number of rules for long time periods (e.g., equivalent to 5 hours of flight time) to draw conclusions with a sufficient amount of confidence. Since it will take approximately 1 hour and 24 minutes to perform this simulation on a 486/33 microcomputer, a lot more running time is needed.

If uncertainty is not a concern, then Algorithm-1 can be used as implemented by PASIM. Using the same inputs as above, except that $8.1e-5$ is used in place of $1.15e-4$ (no uncertainty percent is needed), one finds that the maximum sequential time is almost 0.6 second while the maximum parallel time is under 0.09 second. Hence, based upon just one run, one would conclude that parallel processing of 4,000 rules is fast enough even when 0.1-second sensor updates are used. This does not take any additional communication time into account.

There are some PARSIM (and PASIM) system limitations that should be mentioned: (1) Due to the problem requirements as well as the clock precision on the simulation computer (1/100th of a second), one should not expect reliable timing estimates with fewer than 1,000 rules. (2) The upper limit is 10,000 rules, but if one simulates a rule-base of around 10,000 rules and chooses a maximum of, for example 10 criteria per rule, then on average 50,000 criteria would have to be kept in the *q* array. But, this array is limited to 32,760 locations, so the code will automatically reduce the number of criteria per rule near the end of the generated rule-base to ensure that each rule has at least one criterion. Hence, the actual average number of criteria per rule may be closer to 3 than to 5 because of all the rules that must have only one criterion. It is up to the PARSIM user to determine if this average is realistic. (3) PARSIM run times do not take into account any processor assignment or data communication times. These depend upon both the architecture and hardware being used.

Most of the simulations that were run during this investigation used rule-bases of sizes from 1,000 up to 6,000. The number of distinct actions was arbitrarily input to be the same as the number of rules since this has no effect on the simulation timing at all. (However, if one wishes exact simulation reproducibility, this can be input as a negative number, and the absolute value of this number is used as the initial PRN seed instead of using the system clock.) Two different rule limits were investigated, up to 10

criteria per rule and up to 20 criteria per rule. For example, if a rule is needed to identify a radar according to six pairs of parameter ranges, then up to 12 criteria may be needed in this rule (e.g., $710 \leq f_1 \leq 855$ yields $c_1 = f_1 \geq 710$ and $c_2 = f_1 \leq 855$).

Two different time-steps were also studied: 0.1 and 0.5 second. These were the thresholds used to determine when the simulated maximum sequential or parallel times were good enough, meaning smaller than 0.1 or 0.5 second, respectively. The criterion evaluation times depended upon the algorithm being used. As stated earlier, on average, it was found that Algorithm-1 used 81 microseconds and Algorithm-1u used 115 microseconds to evaluate a single criterion for the 16-MHz Intel 386 processor. That is the available processor closest in clock speed to the 20-MHz transputer.

8. TEST RESULTS AND CONCLUSIONS

Due to the short project time available and the large amounts of running time required, only one or two simulations were made for each rule number and criteria limit combination. All of the conclusions below are based upon these limited cases and should be viewed accordingly. In particular, for maximum rule-base evaluation times close to any desired threshold (e.g., 0.1 or 0.5), more simulations will be necessary.

Based upon the simulations using Algorithm-1 with the PASIM program (no uncertainty addressed), eight parallel processors with a clock speed of 16 MHz or faster were always able to process a rule-base of 4,500 or fewer rules having a maximum of 10 criteria per rule. Single processors at this same speed were unable to do this in under a tenth of a second. The average maximum speed-up was 6.6, using 1 master and 7 rule-processors. A faster single processor, such as the 50-MHz Intel 486, would be able to process the same 4,500 rules within this same time.

Once uncertainty is introduced into the criteria vector, the processing time increases. This was investigated by using Algorithm-1u in the PARSIM code. Here the estimated average time to process a single criterion goes from 81 microseconds to 115 microseconds. In order to process all of the rules within a tenth of a second, with up to 10 criteria per rule, the simulation showed

that the number of rules in the rule-base had to be reduced to 2,500. It appears that up to 2,000 rules, each having up to 20 criteria can be executed in parallel under a tenth of a second (with no extra communication time taken into account). Without parallel processing, not even 1,000 rules could be evaluated. If the sensor update time is increased to a half-second, then up to 6,000 rules with up to 20 criteria each can be processed in parallel (even with a 0.2-second communication time added).

9. REFERENCES

1. James L. Noyes, Artificial Intelligence with Common Lisp: Fundamentals of Symbolic and Numeric Processing, D. C. Heath, Lexington, MA, 1991.
2. Robert R. Trippi and Efraim Turban, "Parallel Processing and OR/MS," Computers & Operations Research, Vol. 18, No. 2, 1991, pp. 199-210.
3. Peter G. Raeth, "An Expert Systems Approach to Decision Support in a Time-Dependent, Data Sampling Environment," in Expert Systems: A Software Methodology for Modern Applications, Edited by Peter G. Raeth, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 170-177.
4. Peter G. Raeth, "Expert Systems in Process Observation and Control," AI Expert, Vol. 5, No. 12, Sep. 1990, pp. 40-45.
5. K. R. Tout and D. J. Evans, "Parallel Forward Chaining Technique with Dynamic Scheduling, for Rule-Based Expert Systems," Parallel Computing, Vol. 18, No. 8, Aug. 1992, pp. 913-930.
6. Bruce P. Lester, The Art of Parallel Programming, Prentice Hall, Englewood Cliffs, NJ, 1993.
7. Stephen K. Park and Keith W. Miller, "Random Number Generators: Good Ones are Hard to Find," Communications of the ACM, Vol. 31, No. 10, Oct. 1988, pp. 1192-1201.
8. William H. Press, et al., Numerical Recipes in Pascal: The Art of Scientific Computing, Cambridge University Press, New York, NY, 1989.